CS432

GPU Accelerated Computing Final Project

**Solving N-body Problem via Barnes Hut**

**Algorithm**

Syed Muhammad Suffwan

[ss03516@st.habib.edu.pk](mailto:ss03516@st.habib.edu.pk)

Syed Muhammad Hasan Haider

[sh03476@st.habib.edu.pk](mailto:sh03476@st.habib.edu.pk)

Submitted to:

Dr. Ayaz ul Hassan Khan

Dec 06, 2020

# Table of Contents

[Table of Contents](#_Toc58183133)

[Abstract](#_Toc58183134)

1. [Problem ………………………………………………………………………………3](#_Toc58183135)
2. [Barnes-Hut Algorithm 4](#_Toc58183136)
3. [Performance & Analysis 5](#_Toc58183137)

[3.1 Barnes-Hut Serial Implementation: 5](#_Toc58183138)

[3.2 Barnes-Hut Naive Implementation: 6](#_Toc58183139)

[3.3 Barnes-Hut Shared Memory Implementation: 7](#_Toc58183140)

[Conclusion 8](#_Toc58183141)

[References 9](#_Toc58183142)

# Abstract

The N-body problem is predicting the behavior of a mass in presence of the other masses and the masses could range. Mathematicians and physicists since the Newtonian era have tried finding different solutions to compute forces and the respective impact of masses on each other. Researchers have tried simulating all-pairs interactions in the past at the computational cost of O(n2). In this project, we have parallelized the Barnes hut algorithm for this problem using CUDA with three different implementations; serial, naïve parallel, and shared memory. The computational cost was reduced to O(n log n) and the performance is discussed further in detail. The GPU machine used to carry out experiments was Tesla Volta 100.

# Problem

Newton’s equations of motion of systems does not incorporate more than two particles so we cannot analyze the impact of more than one mass on a mass in focus directly [1]. The N-body problem is not restricted to systems involving gravitational interactions but it also stretches to other systems at molecular level like atoms, or electrostatically charges ions. In order to calculate the final position of a mass in a system of N masses; we must take initial positions of a particles, velocities, force and mass (kept constant). The challenge of efficiently carrying out the related calculations for N masses is the N-body problem [2]. With efficiency, we mean not only calculating correct results but also lesser computational cost.

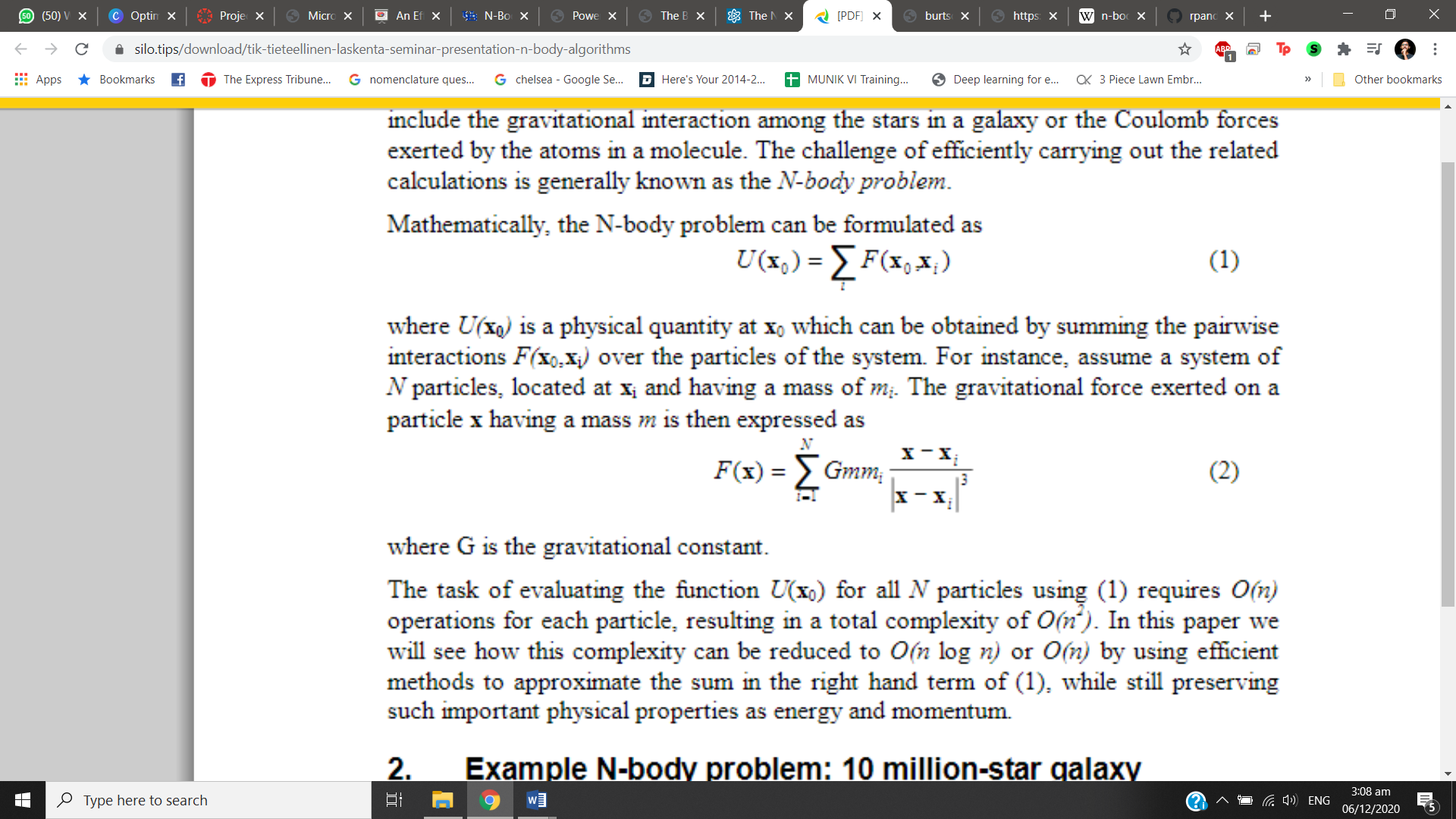


Figure 1: Equation of N-body problem

where U(x0) is a physical quantity at x0 which can be obtained by summing the pair-wise interactions F(x0,xi) over the particles of the system [2]. Assume a system of N particles, located at xi and having a mass of mi. The gravitational force exerted on a particle x having a mass m is then expressed as:

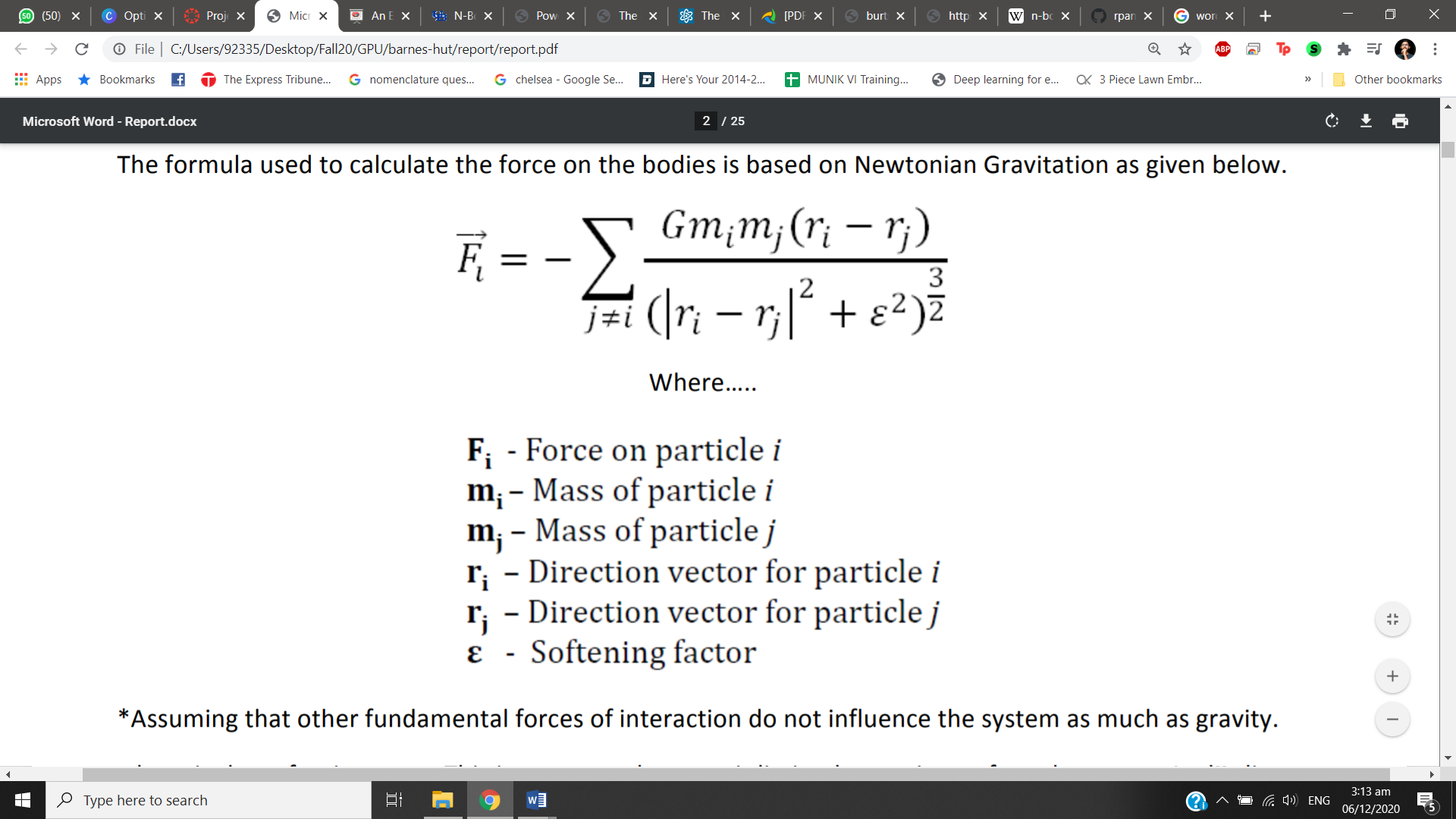


Figure 2: Newton’s gravitational force between two masses

# Barnes-Hut Algorithm

This algorithm is a clever scheme for grouping together bodies that are sufficient nearby. It approximates a particle's dependence on distant data using approximations on their center of mass while the total mass of each body is assumed to be constant [3]. It uses an oct-tree, which is similar to a binary tree, except that each node has 8 children (some of which may be empty). All the n bodies are first recursively divided into groups while storing as node in an octree (3D simulation) where each node is a 3D space. This breakdown continues until each division has a 1 or zero bodies.

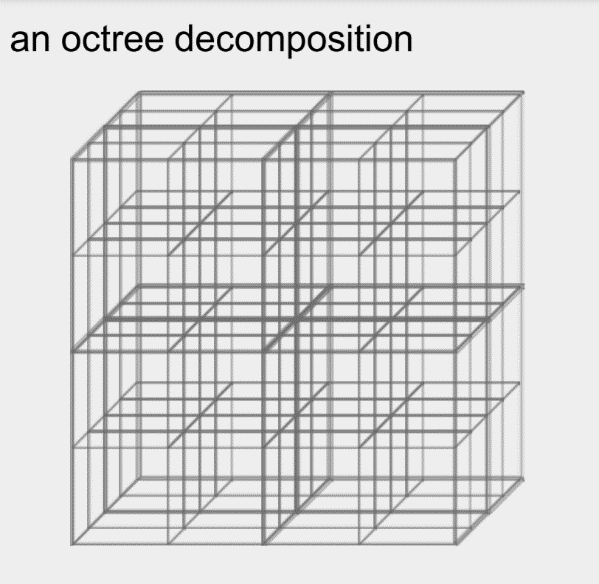




Figure 3: Oct-tree Figure 4: 3-D Oct-tree simulation

Each node here is represents a single body; where internal node stores total mass of its childer bodies. In order to calculate the net force on particular body, we first check that how far is the internal node storing the centre of mass from the body under consideration. If it is far than whole subtree is dealt as 1 body and if its near than we traverse each of the subtree. This is one point where we cut down the compurations by using approximation [4]. This algorithm cuts down the computational cost to O(n log n) where n is the number of bodies involved in the system. Basically, when we implement a Barnes-Hut algorithm, the following steps have to repeated at every time-stemp:

# Performance & Analysis

Following are the results for all the three implementations:

|  |  |  |  |
| --- | --- | --- | --- |
| Nbody | Shared | Serial | Naïve |
| 1000 | 20.6 | 119.098 | 240 |
| 5000 | 22.9 | 3296.125 | 2036 |
| 10000 | 24.6 | 13182.86 | 2191 |
| 50000 | 37.4 | 482310.1 | 2786 |
| 100000 | 54.1 | 2608440 | 3861 |

Table 1: Time results on different size of N-bodies

### 3.1 Barnes-Hut Serial Implementation:

This implementation was carried upon CPU kernel and we due to very high computational cost we could not calculate for n bodies greater than 10,000.

The most time consuming step was building an oct-tree because it was accessing the main memory amply which resulted in longer delays for greater n bodies. The force computation took an order of magnitude longer than the tree construction and this was the motivation to employ parallel approach for calculating force.

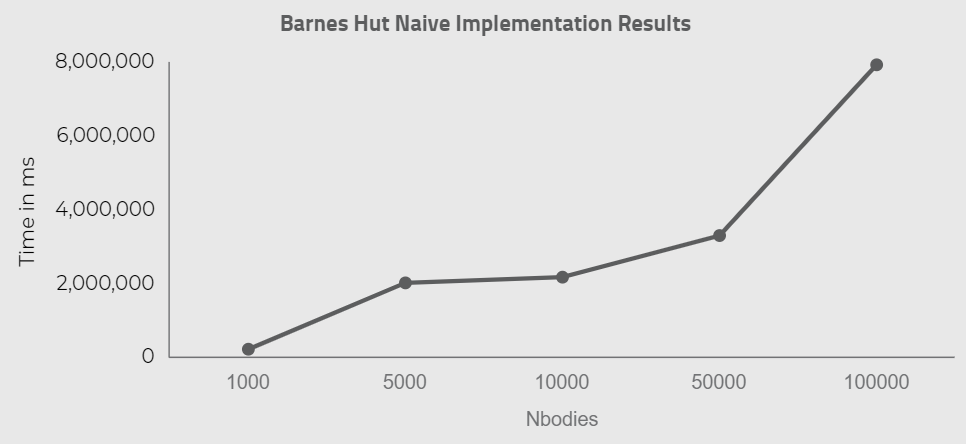


Figure 5: Results for Serial Implementation

### 3.2 Barnes-Hut Naive Implementation:

Contrary to expectation, the Naïve Parallel implementation showed an increase in time taken. Hence, the bottleneck for this implementation was in the updateBodies() kernel. This was observed through the simple print function at the start of each process. Updating body positions took a lot of time. Another approach used was profiling through nvprof which showed that most of the GPU time was taken by the processes of copying data from device to host and vice versa. The reason for this is the use of stream functionality (see figure 1).

Figure 6: Results for Naive Implementation



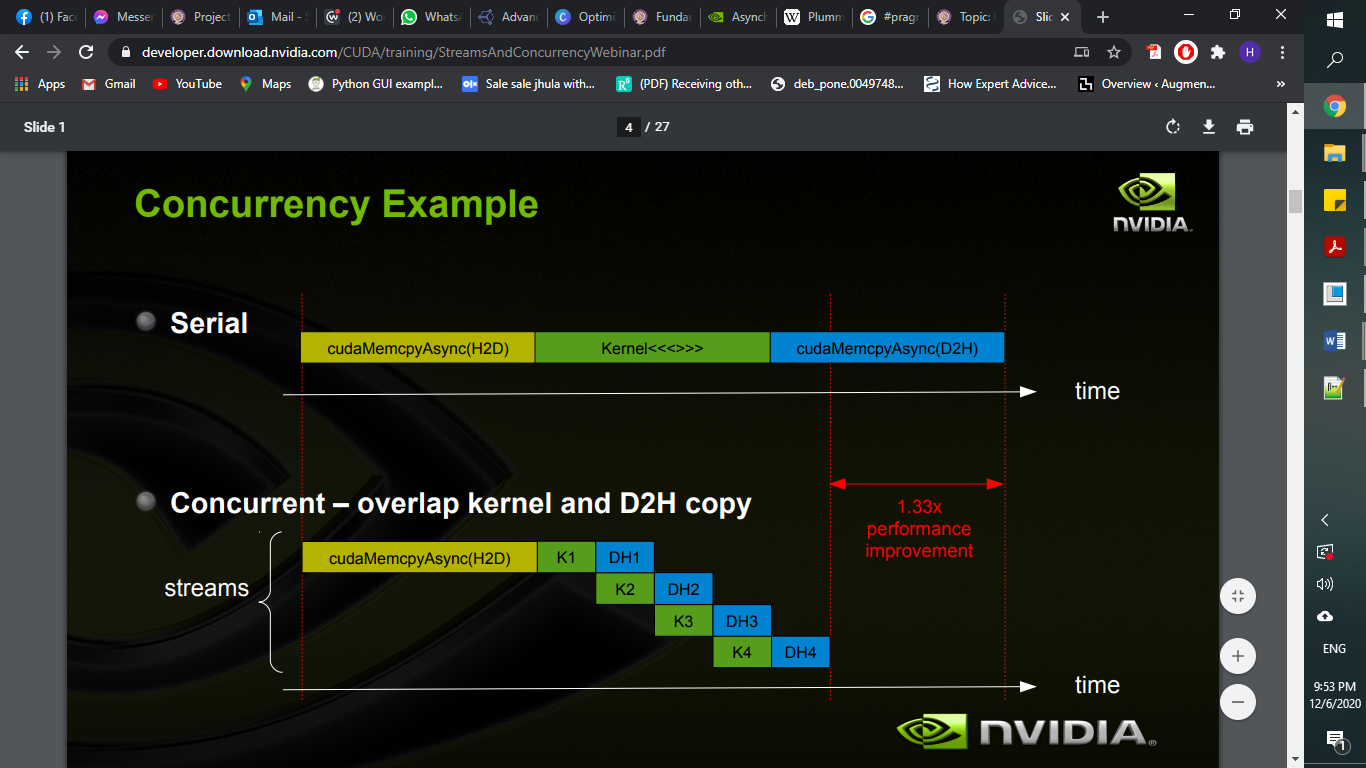


Figure 7: Streams working

### 3.3 Barnes-Hut Shared Memory Implementation:

The greatest time taken was by the kernel which was computing centre of gravity (28% of the total time based on profiling). For example, when we executed this for n bodies size of 1000,000;

Total time (ms) = 534.5

Time taken by COGKernel (ms) = 149.66

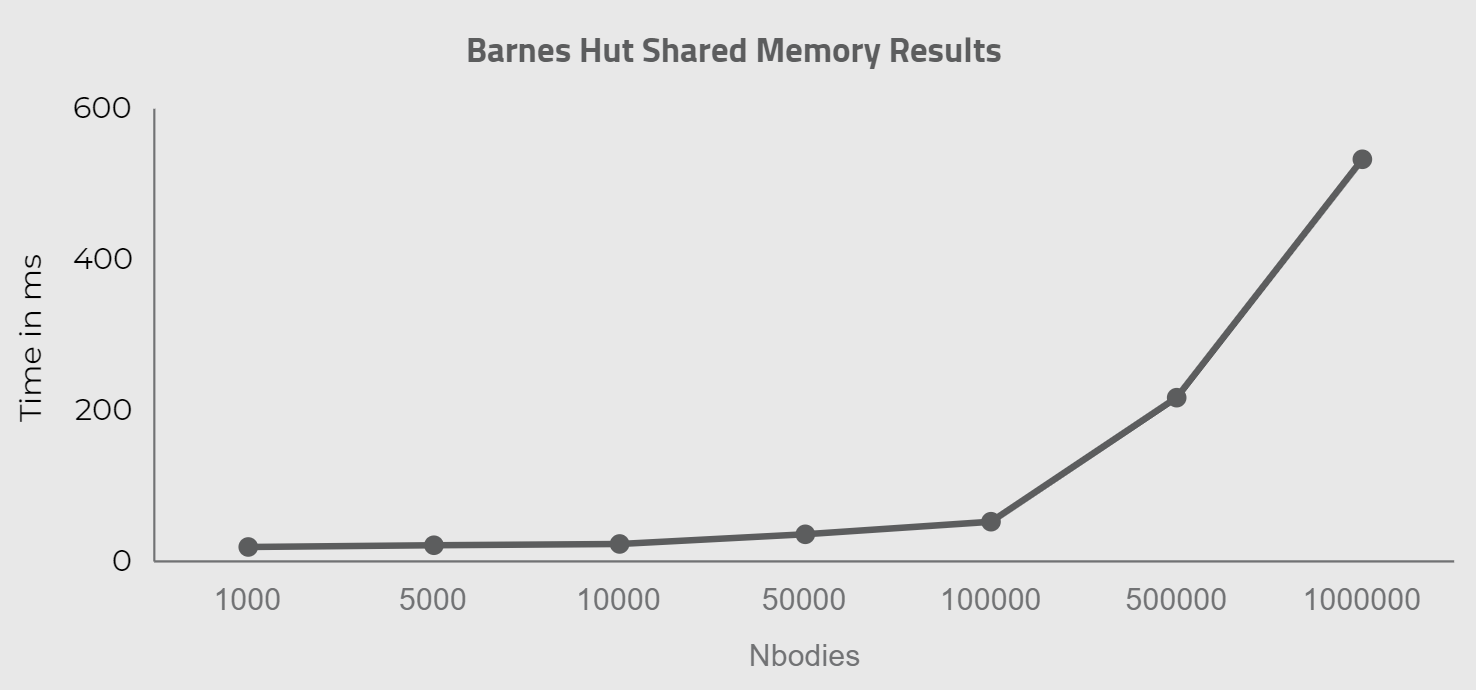


Figure 8: Results for Shared Memory Implementation

# 4. Conclusion

This report explained the Barnes-Hut approach to solve the Nbody problem. Three approaches were used. The first approach was a sequential approach. Second was the Naïve parallel implementation. Lastly, Barnes-Hut algorithm using shared memory was discussed. Performance analysis showed that the Barnes-Hut implementation using shared memory is the fastest one amongst the three approaches. The machine used was **Tesla Volta 100** (see figure 10). You can find the working algorithms in the given repository:

<https://github.com/ss03516/CS432-N-body-using-Barnes-Hut>

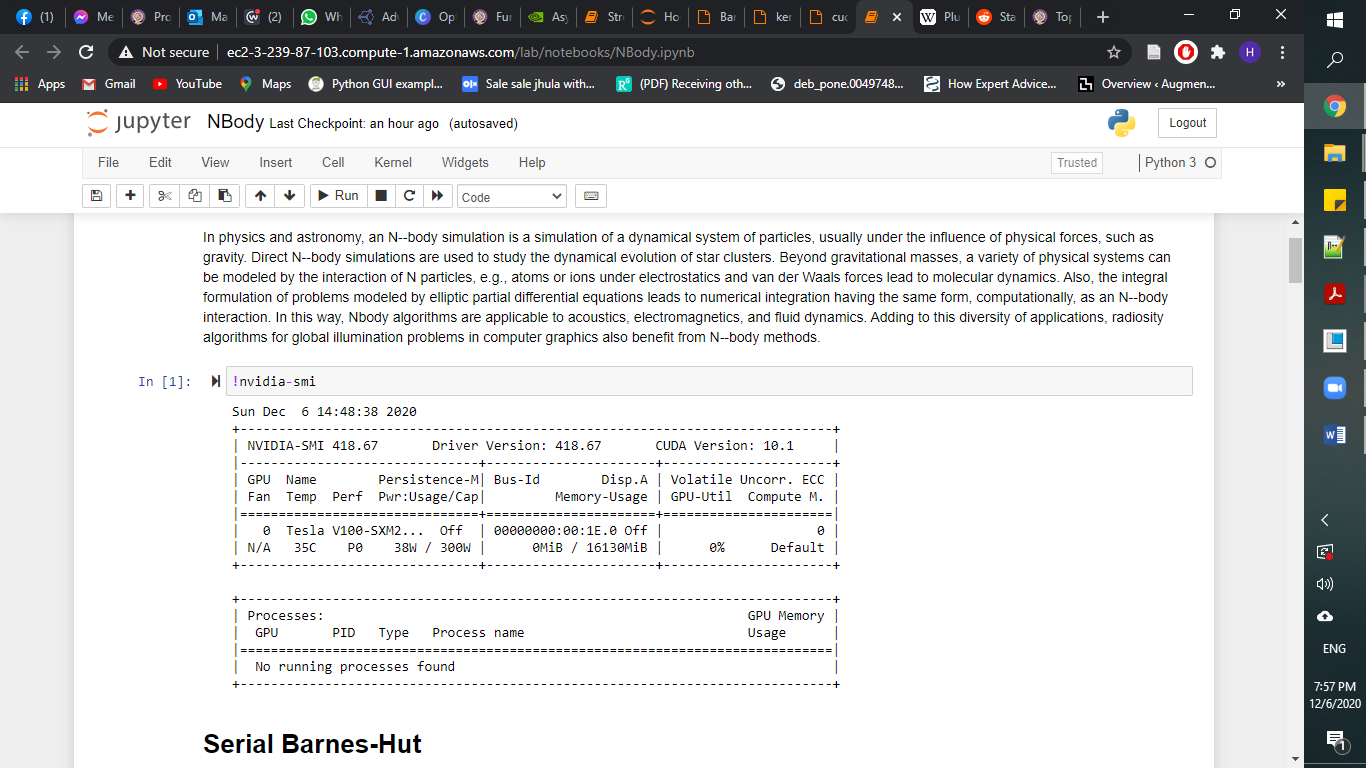


Figure 10: GPU Machine Specifications

# 5. References

[1] G.Leslie, ‘The Numerical Solution of the N-Body Problem”, American Institute of Physics, 1990. [Online]. Available: https://doi.org/10.1063/1.4822898. [Accessed: 24- Nov- 2020].

[2] L. Tik, ‘ N-body Algorithms’, Tancred Lindholm, 2017. [Online]. Available: https://silo.tips/download/tik-tieteellinen-laskenta-seminar-presentation-n-body-algorithms

[3] V. Tom, W. Kevin, ‘The Barnes-Hut Algorithm’, 2003. [Online]. Available: <http://arborjs.org/docs/barnes-hut>

[4] B. Martin, P. Keshav, ‘An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm’. [Online]. Available: <https://iss.oden.utexas.edu/Publications/Papers/burtscher11.pdf>

[5] Figures: <http://www.cs.nyu.edu/courses/spring12/CSCI-GA.3033-012/nbody-problem.pdf>

[6] Codes: <https://drive.google.com/drive/folders/1TFCwgZEC_N1QJpTbyKE_yw617wxf9CLO>

[7]https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf